Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 8

Тема: Асинхронное программирование

Студент: Пивницкий Даниэль Сергеевич
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

Создать приложение, которое будет считывать из стандартного ввода данные фигур, согласно варианту задания, выводить их характеристики на экран и записывать в файл. Фигуры могут задаваться как своими вершинами, так и другими характеристиками (например, координата центра, количество точек и радиус).

Программа должна:

Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;

Программа должна создавать классы, соответствующие введенным данных фигур;

Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки. Например, для буфера размером 10 фигур: **oop_exercise_08 10**

При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;

Обработка должна производиться в отдельном потоке;

Реализовать два обработчика, которые должны обрабатывать данные буфера:

a. Вывод информации о фигурах в буфере на экран;
b. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.

Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.

Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике.

В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;

В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.

Реализовать в основном потоке (main) ожидание обработки буфера в

потоке-обработчике. Т.е. после отправки буфера на обработку основной поток должен ждать, пока поток обработчик выведет данные на экран и запишет в файл.

2. Описание программы

Синхронизация процессов осуществляется посредством двух условных переменных и мьютекса.

3. Набор тестов

test_1

r
hexagon 1 1 1 1 1 1 1 1 1 1 1 1
octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
triangle 1 1 1 1 1 1
q

test_2

r
hexagon 1 1 1 1 1 1 1 1 1 1 1 1
octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
triangle 1 1 0 0 1 0
q

4. Результаты выполнения тестов

Input 'q' for quit, or 'r' to continue

r

hexagon 1 1 1 1 1 1 1 1 1 1 1 1

Added

octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Added

triangle 1 1 1 1 1 1

Added

hexagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

octagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

triangle: (1 1) (1 1) (1 1)

Input 'q' for quit, or 'r' to continue

q

C:\Users\Daniel\source\repos\Lab8\Debug\Lab8.exe (процесс 14948) завершил работу с кодом 0.

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".

Нажмите любую клавишу, чтобы закрыть это окно…

Input 'q' for quit, or 'r' to continue

r

hexagon 1 1 1 1 1 1 1 1 1 1 1 1

Added

octagon 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Added

triangle 1 1 0 0 1 0

Added

hexagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

octagon: (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1) (1 1)

triangle: (1 1) (0 0) (1 0)

Input 'q' for quit, or 'r' to continue

q

C:\Users\Daniel\source\repos\Lab8\Debug\Lab8.exe (процесс 17468) завершил работу с кодом 0.

Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".

Нажмите любую клавишу, чтобы закрыть это окно…

5. Листинг программы

```
//
//  main.cpp
//  lab8
//  Variant 15
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
/*
Создать приложение, которое будет считывать из стандартного ввода данные фигур, согласно варианту задания,
выводить их характеристики на экран и записывать в файл.
Фигуры могут задаваться как своими вершинами, так и другими характеристиками (например, координата центра, количество точек и радиус).
*/
#include <iostream>
#include <vector>
#include <memory>
#include <string>
#include <thread>
#include <mutex>
#include <condition_variable>
#include "factory.h"
#include "subscriber.h"

int main(int argc, char** argv) {
    if (argc != 2) {
        std::cout << "Wrong. \n";
        return 0;
    }
```

```
    int Vecsize = std::atoi(argv[1]);
    std::vector<std::shared_ptr<figures::Figure>> Vec;
    factory::Factory Factory;
    std::condition_variable cv;
    std::condition_variable cv2;
    std::mutex mutex;
    bool done = false;
    char cmd = 'd';
    int in = 1;
    std::vector<std::shared_ptr<Sub>> subs;
    subs.push_back(std::make_shared<Print>());
    subs.push_back(std::make_shared<Log>());
    std::thread subscriber([&]() {
        std::unique_lock<std::mutex> subscriber_lock(mutex);
        while (!done) {
            cv.wait(subscriber_lock);
            if (done) {
                cv2.notify_all();
                break;
            }
            for (unsigned int i = 0; i < subs.size(); ++i) {
                subs[i]->output(Vec);
            }
            in++;
            Vec.resize(0);
            cv2.notify_all();
        }
        });
    while (cmd != 'q') {
        std::cout << "Input 'q' for quit, or 'r' to continue" << std::endl;
        std::cin >> cmd;
        if (cmd != 'q') {
            std::unique_lock<std::mutex> main_lock(mutex);
            for (int i = 0; i < Vecsize; i++) {
                Vec.push_back(Factory.FigureCreate(std::cin));
                std::cout << "Added" << std::endl;
            }
            cv.notify_all();
            cv2.wait(main_lock);
        }
    }
    done = true;
    cv.notify_all();
    subscriber.join();
    return 0;

}


//
//  factory.h
```

```
//  lab8
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef FACTORY_H
#define FACTORY_H

#include <iostream>
#include "figure.h"

namespace factory {

    class Factory {
    public:
        std::shared_ptr<figures::Figure> FigureCreate(std::istream& is) const
{
            std::string type;
            std::cin >> type;
            if (type == "hexagon") {
                                return  std::shared_ptr<figures::Figure>(new
figures::Hexagon(is));
            }
            else if (type == "octagon") {
                                return  std::shared_ptr<figures::Figure>(new
figures::Octagon(is));
            }
            else if (type == "triangle") {
                                return  std::shared_ptr<figures::Figure>(new
figures::Triangle(is));
            }
                  throw  std::logic_error("Wrong.  Figures:  hexagon,  octagon,
triangle");
        }
    };
}
#endif


//
//  figure.h
//  lab8
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef FIGURE_H
```

```
#define FIGURE_H

#include <iostream>
#include <cmath>
#include "point.h"

namespace figures {

    enum FigureType {
        hexagon,
        octagon,
        triangle
    };

    class Figure {
    public:
        virtual std::ostream& print(std::ostream& out) const = 0;
        ~Figure() = default;
    };


    class Hexagon : public Figure {
    public:
        point A, B, C, D, E, F;

        Hexagon() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0, 0 }, E{ 0, 0 }, F{
0, 0 } {}

        explicit Hexagon(std::istream& is) {
            is >> A >> B >> C >> D >> E >> F;
        }

        std::ostream& print(std::ostream& os) const override {
            os << "hexagon: " << A << " " << B << " " << C << " " << D << " "
<< E << " " << F << std::endl;
            return os;
        }

    };


    class Octagon : public Figure {
    public:
        point A, B, C, D, E, F, G, H;

        Octagon() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 }, D{ 0, 0 }, E{ 0, 0 }, F{
0, 0 }, G{ 0, 0 }, H{ 0, 0 }  {}

        explicit Octagon(std::istream& is) {
            is >> A >> B >> C >> D >> E >> F >> G >> H;
        }
```

```
        std::ostream& print(std::ostream& os) const override {
            os << "octagon: " << A << " " << B << " " << C << " " << D << " "
<< E << " " << F << " " << G << " " << H << std::endl;;
            return os;
        }

    };


    class Triangle : public Figure {
    public:
        point A, B, C, D;


        Triangle() : A{ 0, 0 }, B{ 0, 0 }, C{ 0, 0 } {}

        explicit Triangle(std::istream& is) {
            is >> A >> B >> C;
        }

        std::ostream& print(std::ostream& os) const override {
            os << "triangle: " << A << " " << B << " " << C << std::endl;
            return os;
        }
    };

}

#endif


//
//  point.h
//  lab8
//  Variant 15
//  M8o-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef OOP_LAB7_POINT_H
#define OOP_LAB7_POINT_H

#include <iostream>


struct point {
    point() : x(0), y(0) {}
    point(double a, double b) : x(a), y(b) {}
    double x;
```

```
    double y;
};


std::istream& operator>>(std::istream& is, point& p) {
    is >> p.x >> p.y;
    return is;
}


std::ostream& operator<<(std::ostream& os, point p) {
    os << '(' << p.x << ' ' << p.y << ')';
    return os;
}

#endif



//
//  subscriber.h
//  lab8
//  Variant 15
//  М8о-206Б-19
//  Created by Daniel Pivnitskiy on 10.10.2020.
//  github.com/SLAST1
//  Copyright © 2020 Daniel Pivnitskiy. All rights reserved.
//
#ifndef SUBSCRIBERS_H
#define SUBSCRIBERS_H
#include <fstream>

class Sub {
public:
    virtual void output(std::vector<std::shared_ptr<figures::Figure>>& Vec) =
0;
    virtual ~Sub() = default;
};

class Print : public Sub {
public:
     void output(std::vector<std::shared_ptr<figures::Figure>>& Vec) override
{
        for (auto& figure : Vec) {
            figure->print(std::cout);
        }
    }
};

class Log : public Sub {
public:
```

```
    Log() : in(1) {}
     void output(std::vector<std::shared_ptr<figures::Figure>>& Vec) override
{
        std::string filename;
        filename = std::to_string(in);
        filename += ".txt";
        std::ofstream file;
        file.open(filename);
        for (auto& figure : Vec) {
            figure->print(file);
        }
        in++;
    }
private:
    int in;
};



#endif
```

6. Выводы

В ходе выполнения лабораторной работы мною были приобретены начальные навыки работы с асинхронным программированием. Также я научился работать с аргументами программы.

7. Список литературы

1.Руководство по написанию кода на C++ [Электронный ресурс].
URL:https://metanit.com/cpp/tutorial/
Дата обращения: 10.09.2019

2.Документация по C++ [Электронный ресурс]. URL:
https://docs.microsoft.com/ru-ru/cpp
Дата обращения 12.09.2019